

[New chat](#)[Search chats](#)[Library](#)[Atlas](#)[Projects](#)[GPTs](#)[Explore](#)[Image Reader](#)[Your chats](#)[OpenSCAD parser setup](#)[Code update TopoShape che...](#)[Fix TypeError in Python](#)[Zeeman effect and CO2](#)[Update function check](#)[Git ignore .DS\\_Store](#)[Improved checkObjShape fu...](#)[Hull function in OpenSCAD](#)[UK pensioners full pension d...](#)[Gitignore .DS\\_Store files](#)[FreeCAD API event logging](#)[Rewrite function with excepti...](#)[LibreCAD line selection](#)

Keith Sloan  
Free

[Upgrade](#)

IDENTs/NUMBERs). If you want full fidelity, add additional tokens and grammar for arrays.

## Quick example of your desired workflow

- Parse file -> `ast`.
- `mark_top_level_compounds(ast)` -> `compounds, brep_ops = flatten_for_freecad(ast)`.
- For each `compound` :
  - `scad_text = compound.to_scad()` — call OpenSCAD to produce a single mesh/BREP (e.g.,  
`openscad -o out.stl --render -- {stdin or temp file}`).
  - Import the resulting mesh/BREP into FreeCAD as a single object.
- For each `brep_op` :
  - Convert the AST node into FreeCAD primitives / transforms / boolean ops in your importer pipeline.

If you want, I can:

- extend the parser to support arrays ( `[1,2,3]` ) and named arguments as dicts,
- add module definition/call resolution,
- or adapt `collect_brep_ops` into a mapping that directly generates FreeCAD Python API calls (e.g., create `Part` primitives and `Placement` transforms) — I can do that next and show a FreeCAD import stub mapping primitives → FreeCAD calls. Which of these would be most useful to you?

